**Building a Smarter AI-Powered Spam Classifier – Phase 2**



**Name:** Pavan Kumar V

**Reg No:** 513521104034

**Dept:** CSE

**Year:** III

**NM id:** au513521104034

**E-Mail:**

[pavankumar.v1301@gmail.com](mailto:pavankumar.v1301@gmail.com)

**Introduction:**

* In today's digital landscape, where spam messages inundate communication channels, this project embarks on the mission to develop a robust spam classifier. Accurate spam classification is indispensable for users seeking to separate legitimate messages from unwanted content, thereby preserving the reliability of their digital communications.
* Creating an effective spam classifier is a data-driven endeavour that relies on the capabilities of machine learning to analyse historical data and make accurate predictions about whether a message is spam or not. This journey commences with foundational steps, starting with data acquisition and preprocessing to ensure reliable and precise classification.
* This introduction will guide you through the initial steps of the process. We'll explore how to import essential libraries, load the housing dataset, and perform critical preprocessing steps. Data

preprocessing is crucial as it helps clean, format, and prepare the data for further analysis. This includes handling missing values, encoding categorical variables, and ensuring that the data is appropriately scaled.

**Given Dataset:**

* The files contain one message per line. Each line is composed by two columns: v1 contains the label (ham or spam) and v2 contains the raw text.



**Necessary steps to follow:**

**1.Import Libraries:**

* Start by importing the necessary libraries

**Program:**

import numpy as np # linear algebra

import pandas as pd # data processing

from nltk.corpus import stopwords

import nltk

nltk.download('stopwords')

from sklearn.pipeline import Pipeline

from sklearn.naive\_bayes import BernoulliNB , MultinomialNB , GaussianNB

from sklearn.metrics import accuracy\_score

import os

**2.Load the Dataset:**

* Load your dataset into a Pandas Data Frame. You can typically find house price datasets in CSV format, but you can adapt this code to other formats as needed.

**Program:**

df = pd.read\_csv(' E:\spam.csv ')

pd.read()

3. Exploratory Data Analysis (EDA):

* Perform EDA to understand your data better. This includes checking for missing values, exploring the data's statistics and visualizing it to identify patterns.

**Program:**

# Check for missing values

print(df.isnull().sum())

# Explore statistics

print(df.describe())

# Visualize the data (e.g., Bar Charts, histograms, scatter plots, etc.)

**4. Feature Engineering:**

* Depending on your dataset, you may need to create new features or transform existing ones. This can involve one-hot encoding categorical variables, handling date/time data, or scaling numerical features.
* TF-IDF (Term Frequency-Inverse Document Frequency): Convert text data into numerical values representing the importance of words in a document relative to the entire dataset.

**Program:**

tfid\_vect = TfidfVectorizer()

# Extract the tfid representation matrix of the test data.

tfid\_matrix = tfid\_vect.fit\_transform(df['v2'])

print(f"Type :{type(tfid\_matrix)} , Matrix at 0 : {tfid\_matrix[0]} , Shape : {tfid\_matrix.shape}")

5. Split the Data:

* Split your dataset into training and testing sets. This helps you evaluate your model's performance later.

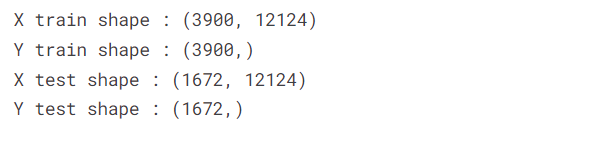
**Program:**

features = df1.drop('v1', axis = 1)

label = df1['v1']

x\_train, x\_test, y\_train , y\_test = train\_test\_split(features , label , test\_size = 0.3)

print(f"X train shape : {x\_train.shape}\nY train shape : {y\_train.shape}\nX test shape : {x\_test.shape}\nY test shape : {y\_test.shape}")



**6. Feature Scaling:**

* Apply feature scaling to normalize your data, ensuring that all features have similar scales. Standardization (scaling to mean=0 and std=1) is a common choice.

**Program:**

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

**Importance of loading and processing dataset:**

* Loading and preprocessing the dataset is an important first step in building any machine learning model. However, it is especially important for house price prediction models, as house price datasets are often complex and noisy.
* By loading and preprocessing the dataset, we can ensure that the machine learning algorithm is able to learn from the data effectively and accurately.

**Challenges involved in loading and preprocessing a spam dataset:**

* There are a number of challenges involved in loading and preprocessing a spam dataset, including

**Handling missing values:**

* Spam datasets often contain missing values, which can be due to a variety of factors, such as human error or incomplete data collection. Common methods for handling missing values include dropping the rows with missing values, imputing the missing values with the mean or median of the feature, or using a more sophisticated method such as multiple imputation.

**Scaling the features:**

* It is often helpful to scale the features before training a machine learning model. This can help to improve the performance of the model and make it more robust to outliers. There are a variety of ways to scale the features, such as min-max scaling and standard scaling.

**Splitting the dataset into training and testing sets:**

* Once the data has been pre-processed, we need to split the dataset into training and testing sets. The training set will be used to train the model, and the testing set will be used to evaluate the performance of the model on unseen data. It is important to split the dataset in a way that is representative of the distribution of the data.

**How to overcome the challenges of loading and preprocessing a spam dataset:**

* There are a number of things that can be done to overcome the challenges of loading and preprocessing a house price dataset, including

**Use a data preprocessing library:**

* There are a number of libraries available that can help with data preprocessing tasks, such as handling missing values, encoding categorical variables, and scaling the features.

**Carefully consider the specific needs of your model:**

* The best way to preprocess the data will depend on the specific machine learning algorithm that you are using. It is important to carefully consider the requirements of the algorithm and to preprocess the data in a way that is compatible with the algorithm.

**1.Loading the dataset:**

* Loading the dataset using machine learning is the process of bringing the data into the machine learning environment so that it can be used to train and evaluate a model.
* The specific steps involved in loading the dataset will vary depending on the machine learning library or framework that is being used. However, there are some general steps that are common to most machine learning frameworks:

1. **Identify the dataset:**

* The first step is to identify the dataset that you want to load. This dataset may be stored in a local file, in a database, or in a cloud storage service.

1. **Load the dataset:**

* Once you have identified the dataset, you need to load it into the machine learning environment. This may involve using a built-in function in the machine learning library, or it may involve writing your own code.

1. **Preprocess the dataset:**

* Once the dataset is loaded into the machine learning environment, you may need to preprocess it before you can start training and evaluating your model. This may involve cleaning the data, transforming the data into a suitable format, and splitting the data into training and test sets.

**Program:**

**import pandas as pd**

**import numpy as np**

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.feature\_extraction.text import TfidfVectorizer**

**from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score**

**from sklearn.naive\_bayes import MultinomialNB**

**from sklearn.svm import SVC**

**from sklearn.ensemble import RandomForestClassifier**

**import xgboost as xgb**

**from nltk.corpus import stopwords**

**import re**

**from sklearn.model\_selection import GridSearchCV**

**from sklearn.pipeline import Pipeline**

**%matplotlib inline**

**import warnings**

**warnings.filterwarnings("ignore")**

**Loading Dataset:**

dataset = pd.read\_csv('E:/spam.csv')

**Data Exploration:**

Dataset:

 **…..**

**2.Preprocessing the dataset:**

* Data preprocessing is the process of cleaning, transforming, and integrating data in order to make it ready for analysis.
* This may involve removing errors and inconsistencies, handling missing values, transforming the data into a consistent format, and scaling the data to a suitable range.

**Some common data preprocessing tasks include:**

**Data cleaning:** This involves identifying and correcting errors and

inconsistencies in the data. For example, this may involve

removing duplicate records, correcting typos, and filling in missing

values.

**Data transformation:** This involves converting the data into a

format that is suitable for the analysis task. For example, this may

involve converting categorical data to numerical data, or scaling

the data to a suitable range.

**Feature engineering:** This involves creating new features from

the existing data. For example, this may involve creating features

that represent interactions between variables, or features that

represent summary statistics of the data.

**Data integration:** This involves combining data from multiple

sources into a single dataset. This may involve resolving

inconsistencies in the data, such as different data formats or

different variable names.

Data preprocessing is an essential step in many data

science projects. By carefully preprocessing the data, data scientists can

improve the accuracy and reliability of their results.

**Program:**

import numpy as np # linear algebra

import pandas as pd # data processing

from nltk.corpus import stopwords

import nltk

nltk.download('stopwords')

from sklearn.pipeline import Pipeline

from sklearn.naive\_bayes import BernoulliNB , MultinomialNB , GaussianNB

from sklearn.metrics import accuracy\_score

import os

for dirname, \_, filenames in os.walk('/kaggle/input'):

for filename in filenames:

print(os.path.join(dirname, filename))

#### Understand the spam collection data!

filepath = '/kaggle/input/sms-spam-collection-dataset/spam.csv'

data\_import = pd.read\_csv(filepath , encoding = 'ISO-8859-1')

data\_import.head()

### Preprocessing!

df = data\_import.drop(['Unnamed: 2' , 'Unnamed: 3' , 'Unnamed: 4'] , axis = 1)

df.head()

**Removing stopwords**

sw = stopwords.words('english')

def stopword(text) :

txt = [word.lower() for word in text.split() if word.lower() not in sw]

return txt

df['v2'] = df['v2'].apply(stopword)

df.head()

### Stemming

from nltk.stem.snowball import SnowballStemmer

ss = SnowballStemmer("english")

def stemming(text) :

text = [ss.stem(word) for word in text if word.split()]

return "".join(text)

df['v2'] = df['v2'].apply(stemming)

df.head()

**Feature Engineering**

from sklearn.feature\_extraction.text import TfidfVectorizer

tfid\_vect = TfidfVectorizer()

tfid\_matrix = tfid\_vect.fit\_transform(df['v2'])

print(f"Type :{type(tfid\_matrix)} , Matrix at 0 : {tfid\_matrix[0]} , Shape : {tfid\_matrix.shape}")

**Data Splitting**

**features = df1.drop('v1' , axis = 1)**

**label = df1['v1']**

**x\_train , x\_test , y\_train , y\_test = train\_test\_split(features , label , test\_size = 0.3)**

**print(f"X train shape : {x\_train.shape}\nY train shape : {y\_train.shape}\nX test shape : {x\_test.shape}\nY test shape : {y\_test.shape}")**

**Preprocessing and Feature Scaling using Pipeline**

**ber\_pipe = Pipeline(steps = [**

**( 'ber\_model' , BernoulliNB())**

**])**

**multi\_pipe = Pipeline(steps = [**

**('multi\_model' , MultinomialNB())**

**])**

**guass\_pipe = Pipeline(steps = [**

**('guass\_model' , GaussianNB())**

**])**

**def model\_evaluation(model) :**

**model.fit(x\_train , y\_train)**

**y\_pred\_model = model.predict(x\_test)**

**acc\_score = accuracy\_score(y\_test , y\_pred\_model)**

**print(f"Accuracy Score of {model[0]} : {acc\_score}")**

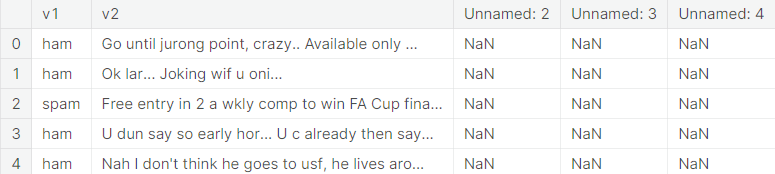
**model\_evaluation(ber\_pipe)**

**model\_evaluation(multi\_pipe)**

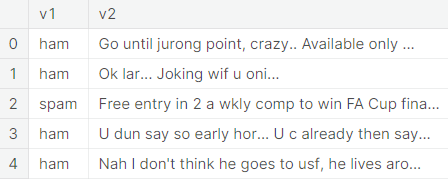
**model\_evaluation(guass\_pipe)**

**Output:**

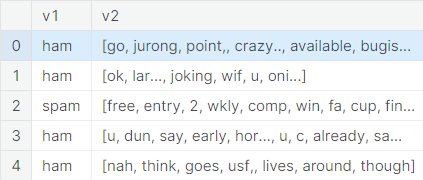
Load the dataset



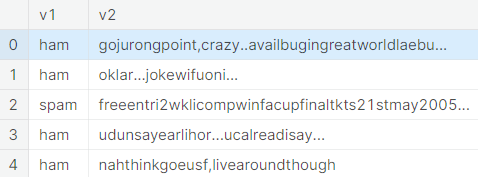
Preprocessing



Stop words



Stemming



Feature Extraction

Type :<class 'scipy.sparse.\_csr.csr\_matrix'> , Matrix at 0 : (0, 1827) 0.5056391989470028

(0, 1030) 0.5056391989470028

(0, 2166) 0.48268727087494234

(0, 3635) 0.5056391989470028 , Shape : (5572, 12124)

Data splitting

X train shape : (3900, 12124)

Y train shape : (3900,)

X test shape : (1672, 12124)

Y test shape : (1672,)

Preprocessing and Feature Scaling using Pipeline

Accuracy Score of BernoulliNB() : 0.8947368421052632

Accuracy Score of MultinomialNB() : 0.9204545454545454

Accuracy Score of GaussianNB() : 0.46411483253588515

#### Multinomial NB is performing better than other models.

**Conclusion:** Preprocessing for a Robust Spam Classifier

In this phase of our project, we have diligently prepared and transformed our data to lay the foundation for building a robust spam classifier. Through careful data cleaning, text preprocessing, and feature engineering, we have strived to optimize the quality of our dataset, ensuring that our machine learning models can learn effectively.

By selecting and engineering relevant features, we have equipped our classifier to capture the nuanced characteristics of spam and non-spam messages. This meticulous preparation is fundamental to achieving high precision, recall, and overall model accuracy.

As we move forward into the modeling and evaluation stages, we do so with the confidence that our data is primed for the challenges of classifying and mitigating spam. The success of our spam classifier hinges on the excellence of our data preprocessing efforts, and we are now well-prepared to tackle the next stages of this exciting project.